Communication on the Paragon*

David Greenberg [1] Barney Maccabe [2] Kevin S. McCurley [1]
Rolf Riesen [2] Stephen Wheat [2]

October 15, 1993

Abstract

In this note we describe the results of some tests of the message-passing performance of the Intel Paragon. These tests have been carried out under both the Intel-supplied OSF/1 operating system with an NX library, and also under an operating system called SUNMOS (Sandia UNM Operating System). For comparison with the previous generation of Intel machines, we have also included the results on the Intel Touchstone Delta. The source code used for these tests is identical for all systems. As a result of these tests, we can conclude that SUNMOS demonstrates that the Intel Paragon hardware is capable of very high bandwidth communication, and that the message coprocessor on Paragon nodes can be used to give quite respectable latencies. Further tuning can be expected to yield even better performance.

Introduction

One of the primary impediments to achieving high efficiency on parallel machines is the cost of interprocessor communication. The Paragon attempts to mitigate the communication costs by employing a mesh architecture which allows very fast communication links (200Mbyte/sec/channel, although our machine is currently running in slow streaming mode, or 175Mbytes/sec). Further architectural support is supplied via wormhole routing which, in theory, allows delivery time to be independent of the distance the message travels in the network. It has become clear to many researchers that a major cost in communication is the time spent preparing a message to be sent over the links and reassembling it when it arrives. The Paragon promises two hardware features to aid in reducing this cost, line transmission units and message co-processors.

We have experimented to see whether it is possible to achieve end-to-end message passing which achieves the high speeds supported by the links. In particular we compare the

*Supported in part by the U.S. Department of Energy under contract DE-AC04-76DP00789.
[1]Organization 1423, Sandia National Laboratories, Albuquerque, NM, 87185
[2]Organization 1424, Sandia National Laboratories, Albuquerque, NM, 87185

performance of the Intel NX message passing library under the OSF operating system with a locally developed operating system called SUNMOS (Sandia UNM Operating System). We have run a number of tests on the paragon, but here we will report on two of these:

- the Intel SAT comtest
- a set of "bucket brigade" tests suggested by R. Littlefield [1].

The current implementation of SUNMOS includes message passing routines to support the nCUBE VERTEX message passing routines (`nread/nwrite`) as well as a subset of the Intel NX message passing routines. In addition, a rudimentary implementation of the Intel NX-compatible global operations are supported. All of these are implemented in terms of the underlying message passing mechanism of SUNMOS, which uses somewhat different semantics (closer to that used by nCUBE's VERTEX system).

First, the user code running on a node "owns" control of the communication channels in and out of a node. Unlike NX/OSF, there is no competition for this resource with the kernel or other users. Since SUNMOS is a single-tasking operating system, when a user wants to send a message, the user knows that they can go ahead and send it out (unless part of the channel to the destination is blocked by other messages, in which case hardware will queue them through the channels). On the sending side, when a message is sent, it immediately goes out to the destination (or blocks until the message can be started). Each node has a communication buffer space used for receiving messages. If a receive has already been posted, then when a message comes in, it goes straight to the user buffer. If a receive has not been posted, then it is stored in the communication space of the node for later retrieval. The size of the communication space is configurable when the job is loaded on the compute nodes. Because of various memory alignment requirements, the message may be copied on either the sending or receiving side if the buffers are unaligned. This may seem insignificant, but a standard `memcpy()` runs slower than sending a message!

Another major difference between SUNMOS and NX/OSF arises from the fact that messages are not packetized under SUNMOS, but NX/OSF currently packetizes messages in packets of not more than 1792 bytes[2]. From the limited information that was available, we were unable to completely understand the protocols used by NX/OSF for message passing, but it appears that each node sets aside a constant (configurable) number of buffers for receiving packets from each other node, and some additional space that it assigns to other incoming packets. Some form of flow control is used to avoid overflowing these buffers.

In previous studies on the Delta we have shown that the speed of collective communication routines can depend greatly on the algorithm used. The large size of Sandia's Paragon system (16x118 nodes) should make these differences even clearer. Messages will potentially have to travel long distances and the congestion at the middle of the machine could be extreme. We expect that the low startup overheads of SUNMOS will also affect the choice of algorithm. We plan to report on experiments with algorithms for global operations in a future paper.

Since both operating systems used in our tests are under continuing development, this is necessarily a snapshot of the current status (as of this writing, on September 23, 1993) that will be quickly out of date. The version of OSF that was tested is referred to as Transmittal

11, and is a beta version that followed official Release 1.0c. The version of NX used on the Delta is 1.5 (the slow production kernel).

The Intel System Acceptance Test (SAT)

Intel ships a set of tests known as the System Acceptance Tests (SAT, in `/usr/lib/sat`) for the Paragon. Included in this test is a program called comtest that is designed to test communication. We ran this test on the Paragon under both SUNMOS and OSF, with the intent to measure the performance of message passing. We also attempted to run this test on the Delta, with limited success.

The SAT comtest currently consists of a number of tests. Some of the tests are *not* scalable, placing them in the category of stress tests rather than reflective of what a well-planned application might use. In particular, the "fan-in" test requires every node to send a message to node 0, which will deadlock node 0 under OSF if it is run on too many nodes, and can consume a great deal of comm space on node 0 under SUNMOS. For example, buffering for the fan-in test requires that node 0 be prepared to receive a number of messages from every other node. Under SUNMOS, this can be accomplished by setting a comm space aside that is big enough to accomodate all of the messages, but it's a pretty silly use of memory. Unfortunately, we could not figure out how to run this under the NX buffering scheme used on the Paragon under OSF with a large number of nodes - it caused node 0 to exhaust it's buffer space, no matter how big we set it.

Machine	OS	unforced csend	unforced isend	median alpha
Delta	NX 1.5	75	78	84
Paragon	OSF T11	61	52	105
Paragon	SUNMOS	33	33	56
Paragon	SUNMOS -p1	24	24	40

Table 1: Latency figures from Intel's SAT comtest, in microseconds. Timings under OSF and NX are for unforced message types. SUNMOS -p1 indicates the use of the message coprocessor.

The tests in the SAT comtest are: ping-pong, ring, broadcast, bisection, random, latency, all to all, exchange, fan-in, and corner-to-corner. We ran these tests on the Paragon under OSF and SUNMOS, and also on the Touchstone Delta at Caltech. We found that the fan-in would not run correctly on a large number of nodes under any circumstance, and the ring and all-to-all tests were extremely slow on the Delta (to the point where it was hard to tell if they were making progress).

In this short note, we give the results from the latency and ping-pong test, primarily because seem to most accurately measure latency and bandwidth. We found that the ping-pong

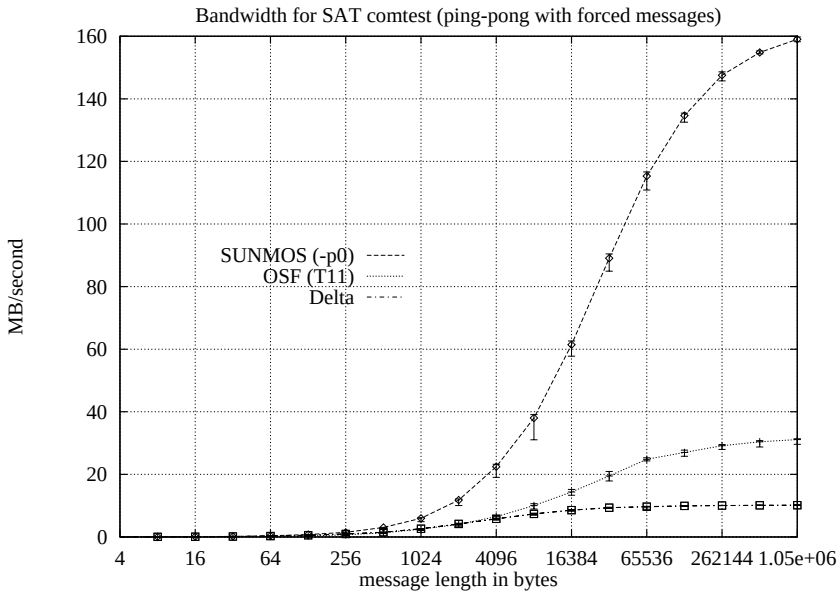

Figure 1: Bandwidth using ping-pong from Intel SAT comtest. Shown are figures for SUNMOS on the Paragon, OSF T11 on the Paragon, and NX 1.5 on the Delta.

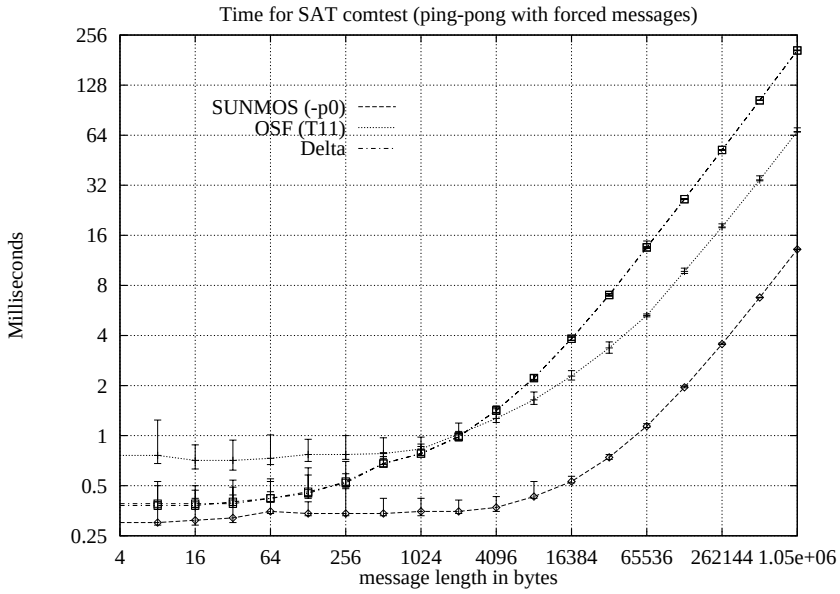

Figure 2: Timing of ping-pong test from Intel SAT comtest. Shown are figures for SUNMOS on the Paragon, OSF T11 on the Paragon, and NX 1.5 on the Delta.

test ran reliably on all three combinations (Delta, Paragon OSF, and Paragon SUNMOS). Moreover, it gives a better indication of the types of bandwith figures that can be obtained in real applications than many of the other tests. The ping-pong test uses two active nodes, 0 and $n-1$. Node 0 sends out a number of messages to node $n-1$, who then receives them and returns them immediately. The roundtrip time for node 0 to send out and receive back all of the messages is calculated. [1]

The bandwidth figures from this test are shown in Figures 1 and 2. Notable in these results is the fact that SUNMOS reaches a bandwidth of over 100 megabytes/second for messages of only 64K, and peaks at close to 160 megabytes/second. This compares well with the 175 megabytes per second peak rating of the hardware in slow streaming mode.

We should mention that early tests with the ping-pong routine under SUNMOS revealed a deficiency in the `memcpy` routine. When we first ran the test, we got quite unpredictable results, with some runs producing a bandwidth of 160 megabytes per second, and some running at only 30 megabytes/second, with no predicatable behaviour. After close examination, we discovered that there were slight timing differences that could occur during the load that would influence whether a receive for a message was posted before it actually arrived at the node. In the case when the receive was posted, SUNMOS placed it directly into user memory with Direct Memory Access (DMA). When the receive had not been posted, SUNMOS allocated communication buffer space and routed the message directly to this location. When the receive was later posted by the user, SUNMOS used `memcpy` to copy it from comm space to user space, and it turned out that the first implementation of `memcpy` was very slow. Some minor effort at optimizing this routine eliminated the huge difference, but graphically illustrated the importance of paying attention to memory usage when passing large messages.

In running the SAT comtest, we also measured latency figures under all three machines. Latency can be measured in a variety of ways. For example, consider the situation where two nodes exchange zero-length messages:

node 0		node 1
send	$\rightarrow$	receive
receive	$\leftarrow$	send

If the round trip time is measured for this and used as a means of measuring latency, then we should keep in mind that the time to initiate the send is overlapped with the time to initiate the receive, resulting in a smaller overall time. We shall not be concerned with this issue here, because our goal is to run *the same code* under two different operating systems. The figures given here correspond to the "loopback" α discussed in [1]. Comparisons to latency figures on other machines require a more in-depth analysis, since the term "latency" is used to mean several different things, and are strongly dependent on the assumptions made for the software model.

The results of the SAT latency comtest are given in Table 1. The time to complete a send and receive operation is shown, as well as the loopback α. Note that actual latencies

[1] One of the authors likens this to Australian-rules ping pong, where the server is allowed to serve five balls at once before waiting to let the other player hit one back.

for SUNMOS are about 15% better if the code is written to use the native `nsend/nrecv` calls instead of Intel NX calls. The differences between the respective libraries are minor, but compatibility with NX behavior imposes some overhead. SUNMOS provides the option to use the message coprocessor, and latency figures are given for both (all other figures in this paper do not use the coprocessor). Experiments are planned for the future to determine the best use under SUNMOS of the second processor of the Paragon GP node.

It should also be noted that the results from OSF are unpredictable, possibly due to difficulties in eliminating measurement of time to page various pieces of code. We used the -plk option on loading and ran numerous iterations, but we still repeatedly observed maximum and minimum times for alpha that range from 101 to over 12000 microseconds. The latter figure was a maximum latency, whereas the median was usually around 110 microseconds. It appears that the first message is extremely expensive, for reasons not understood by us. Experience shows that timing codes on the Paragon under OSF is a difficult and haphazard process.

Littlefield's bucket brigade test

R. Littlefield of Pacific Northwest Labs has argued persuasively that a good measure of message passing performance can be determined from what he refers to as a "bucket brigade", where processors pass data around a ring embedded into the mesh. In a paper presented at the 1992 Intel Users' group Meeting[1], he presented the performance of the Delta and iPSC/860 under a variety of techniques. There are a number of protocols that an be used to pass messages around a ring, depending on

- whether a node does a simultaneous send and receive,
- whether "forced" message types are used with handshaking,
- whether blocking or nonblocking messages are used.

Eight variations based on combinations of these factors are described in Appendix A. In order to eliminate effects from contention, messages were passed around a ring of eight processors illustrated in Figure 5, using each of the eight methods.

Timings were made of the time to pass a message all the way around the ring, after which the time was divided by the number of nodes to calculate the time to pass on one message and receive another. Space constraints prevent us from showing all of the results here, but two of the timings for these are given in Figures 4 and 3. In every case, SUNMOS was at least a factor of three faster than OSF across the board for all message sizes, and sometimes a factor of six. Note that the timings for messages of zero length would correspond to the "shift" alpha of [1].

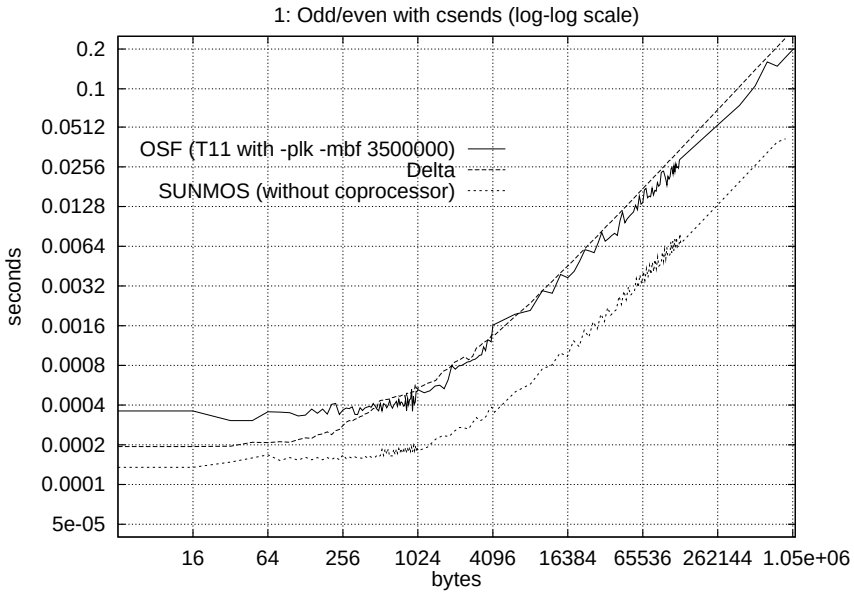

Figure 3: Littlefield Bucket Brigade, Method 1, under SUNMOS, OSF (T11), and Delta NX. The even numbered nodes first call `csend()`, and then call `crecv()`. The odd numbered nodes first call `crecv()` and then `csend()`.

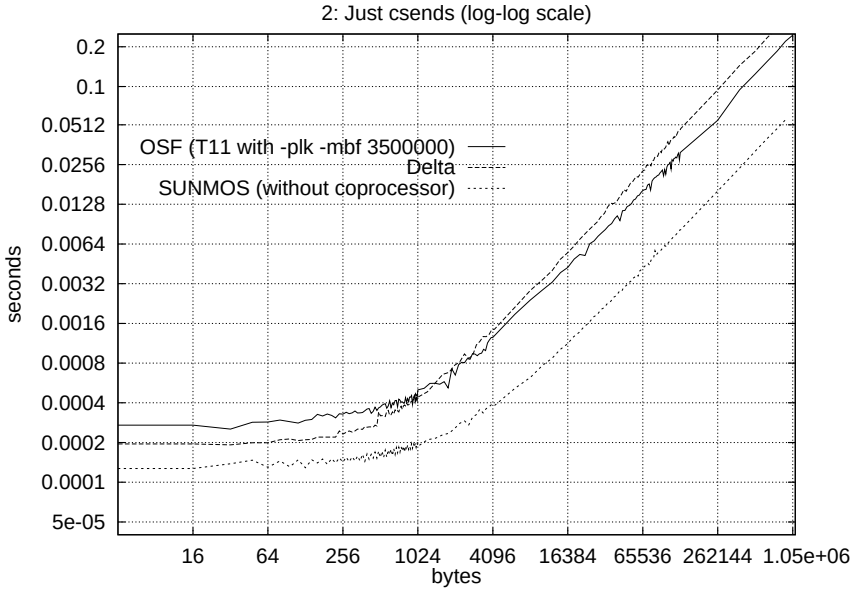

Figure 4: Littlefield Bucket Brigade, Method 2, under SUNMOS, OSF (T11), and Delta NX. Each node first calls `csend()`, and then `crecv()`.

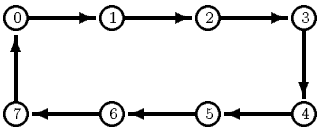

Figure 5: Ring path for messages in Littlefield ring test.

Conclusions

From these tests, we can conclude that the Intel Paragon hardware is capable of very high bandwidth communications between nodes. It is also apparent that the second processor can be exploited to give considerably lower latency figures. Both operating systems (NX/OSF and SUNMOS) will continue to evolve and improve, but the assumptions made in a single tasking operating system such as SUNMOS make it easier to achieve high performance.

References

[1] Richard J. Littlefield, Characterizing and Tuning Communications Performance on the Touchstone DELTA and iPSC/860 (extended abstract). Proceedings of the 1992 Intel Supercomputer Users' Group Conference.

[2] Paragon OSF/1 User's Guide.